WebProfiler: cooperative diagnosis of Web failures

Sharad Agarwal
sagarwal@microsoft.com
Microsoft Research, Redmond

Nikitas Liogkas
nikitas@acm.org
University of California, Los Angeles

Prashanth Mohan
prmohan@microsoft.com
Microsoft Research, India

Venkat Padmanabhan
padmanab@microsoft.com
Microsoft Research, India

October 2008
Technical Report
MSR-TR-2008-45

Despite tremendous growth in the importance and reach of the Web, users have little recourse when a Web page fails to load. Web browsers provide little feedback on such failures, and suggest re-checking the URL or the machine's network settings. Hence, users are often unable to diagnose Web access problems, and resort to haphazardly modifying their settings or simply trying again later. We advocate a client-based collaborative approach for diagnosing Web browsing failures. Our system, WebProfiler, leverages end-host cooperation to pool together observations on the success or failure of Web accesses from multiple vantage points. These are fed into a simple, collaborative blame attribution algorithm. Our evaluation on a controlled testbed shows WebProfiler can accurately diagnose 3.6 times as many failures than possible from a single client's perspective. We present the design and prototype implementation of WebProfiler for an enterprise network.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

1 Introduction

Internet Explorer cannot display the webpage
Most likely causes:

- *You are not connected to the Internet.*
- *The website is encountering problems.*
- *There might be a typing error in the address.*

— IE-7 error page

Despite the tremendous growth of the Web and our reliance on it, a surprisingly large number of attempts to access websites still fail today [18]. The popularity of the Web is fueling an ever increasing number of users which in turn places additional load on HTTP servers and other Web infrastructure. More complex Web applications, such as social networking websites, require server resources beyond that of serving traditional static web pages. The spread of the Internet to various corners of the World is increasing the complexity of Internet infrastructure such as the routing, DNS and CDN systems. Users now expect Web access from more diverse locations, such as mobile phones and the workplace, which typically employ infrastructure such as firewalls and HTTP proxies to filter malicious content, restrict user access and adapt content to different devices. As a result, users do see web failures and they are increasingly difficult to automatically diagnose.

When a failure does occur, users are typically presented with vague and generic information about the cause. The less-than-helpful error message from Microsoft Internet Explorer 7 shown above is by no means specific to that browser — Firefox 2's error message is equally useless. Users may try to reload the page repeatedly, modify their machine or browser configuration, or simply try again later. Yet, depending on the cause, none of these actions may resolve the problem, leaving the user clueless and frustrated.

There are existing techniques for diagnosing Web failures. **HTTP server logs** can be analyzed [16] by website administrators to look for requests that generated HTTP errors. However, these logs do not contain access attempts that did not even reach the website (e.g., due to DNS failures, HTTP proxy failures). **HTTP proxy or edge router logs** can be analyzed by enterprise administrators — medium to large corporations typically use HTTP caching proxies to improve performance and filter harmful web content. These logs include details such as the client IP address, server IP address, URL accessed and HTTP success/error code. However, again, logs collected at the network edge do not include access attempts that failed to reach it (e.g., due to DNS failures, internal routing failures). Recently proposed techniques [9] for constructing **inference graphs of multi-level dependencies** in enterprise networks use packet traces to probabilistically infer the network components involved in a transaction and identify the culprit if a large number of requests fail. Such systems rely on probabilistic techniques to separate out multiple simultaneous network flows into transactions related to the same application-level action, and are impeded when end-to-end encryption such as IPSec is used.

We present **WebProfiler**, which employs end-host *cooperation* to pool together observations of Web access failures and successes from *multiple vantage points*. WebProfiler exploits clients that use different network elements (e.g. DNS servers, HTTP caches) in a *collaborative blame attribution* algorithm to identify those network elements that are often involved in failures rather than successes. Our *distributed* system relies on a *lightweight* Web browser plug-in on each client that monitors the success and failure of any web transaction and shares this with the WebProfiler system. The use of a lightweight application layer plug-in for monitoring results in a system that works even with IPSec encryption, detects *all* client failures, does not require packet sniffing and can be incrementally deployed via software updates.

While the concepts behind WebProfiler are applicable to other networks such as residential access and the Internet in general, we focus on enterprises in our implementation and evaluation. In doing so, we are

able to present the cooperative system design and collaborative diagnosis algorithm in detail without having to address system scalability beyond large enterprises (e.g. 100,000 employees). In terms of privacy, even though our approach detects more errors than in router or proxy logs, the type of data we collect is similar to what enterprise administrators can see in such logs. Furthermore, users do not directly interact with the data, but instead see the names of network components that our diagnosis tool blames for failures. We focus on enterprises without loss of generality in diagnosis. In fact, typical enterprise networks increase diagnosis complexity due to additional network components that can contribute to Web failures — internal vs. external DNS servers, NetBios or WINS resolution, HTTP proxies, internal vs. external websites, etc.

Our novel contributions include an architecture for sharing Web success/failure observations across a diverse set of clients, a simple blame attribution algorithm that identifies the most likely suspect(s) responsible for a failure, a working prototype and its experimental evaluation. Experiments with 25 clients over 13 days on a controlled testbed demonstrate that WebProfiler can accurately diagnose 3.6 times as many failures than from a single client's perspective.

2 Motivation

2.1 Nature of Web failures

Visiting a website through a browser can involve a series of network transactions, depending on the network configuration (§ 5.5 defines some of these protocols) :

- lookup HTTP proxy name via WPAD
- lookup proxy IP addr. via WINS or internal DNS server
- issue HTTP request to proxy
- proxy looks up website addr. via external DNS server
- proxy issues HTTP request to website IP address
- website's or CDN replica's HTTP server responds
- firewall or proxy parses, filters content for client

Some failures can be trivially diagnosed. For example, the Ethernet cable is unplugged; the local DNS server has crashed and is not responding to any lookups. In such cases, the cause can be identified from a single client's perspective [7].

Failures that are not fail-stop are much harder to diagnose. For example, a client's local subnet switch is overloaded and is dropping 20% of packets or flows; one of the CDN servers has crashed and so only *some* of the website accesses fail. Without additional information from beyond one client, such as the status of accesses from other subnets or from clients with other DNS servers that point to different CDN replicas, the failure is difficult to diagnose.

Failures that have multiple culprits are also much harder to diagnose. For example, a website denies accesses from a particular IP address (for a corporate proxy or public library's NAT router; because the site's IDS has mistaken it to be a bot due to the large number of requests from it). With respect to resolving the client's problem, the combination of this website and this proxy is to blame — the client can either continue to use this proxy for other websites, or change the proxy and then access this website.

In our prior Web failure measurement study [18] involving a diverse set of 134 clients in academic and corporate, broadband and dial-up networks, we observed a range of problems that impacted Web access.

Many failures could not be diagnosed from a single client's vantage point, for example where a website was accessible from some clients but not others.

2.2 Why collaborative diagnosis?

To overcome limited visibility, we can leverage multiple clients in the hope that they have different yet overlapping views of the network. For example, a client that fails to establish an HTTP session with a web server cannot reliably determine if the HTTP server is down, or is refusing connections from the client IP address (or that of its proxy), or was pointed to a stale replica of the website by DNS. However, information from another client on the same subnet that uses a different DNS server and gets a different but working replica of the website can help diagnose and solve the problem.

Many networks have clients that have some network elements in common but differ on others. For example, medium to large enterprise networks typically have multiple subnets, multiple internal DNS servers and multiple HTTP proxies. Collaborative diagnosis across such diverse clients would provide an interwoven mesh of observations on Web accesses, helping to identify the most likely suspects responsible for the observed failures.

2.3 Why client cooperation?

Compared to approaches that analyze server logs or packet traces, client cooperation has several advantages. By collecting information on Web successes and failures directly from the client, *all* types of errors are captured, even those that are not recorded in proxy or server logs. User behavior can also be directly captured — did the Web page not load because the user hit "stop" early in the transaction? Users prematurely aborting transactions should not be mistaken for failures.

Capturing information directly from the client application has four main advantages that greatly reduce complexity compared to prior techniques that rely on packet traces.

- Parsers for the various protocols in packet traces are not needed (HTTP, DNS, WINS, TCP, etc.) — the application layer that issues the protocol commands can directly provide this information as well as resulting error codes.
- The Web browser maintains state about individual network transactions associated with the same URL access, which we can directly obtain. In contrast, packet traces will show multiple overlapping network transactions and reconstructing the original application transaction is not trivial. For example, a single attempt to access http://www.cnn.com/ can trigger several DNS lookups and HTTP transactions (different servers for HTML, images, advertisements). If a user has multiple browser windows open, or multiple applications, this can be even harder.
- Compared to packet traces, the amount of information that is needed from the application layer is extremely small and hence places almost no load on the client.
- Application layer capture is agnostic to network layer encryption such as IPSec.

This approach allows incremental deployment via software updates. Furthermore, greater the number and diversity of contributing clients, the more accurate the diagnosis is likely to be.

3 WebProfiler Architecture

Clients cooperating in WebProfiler collect fine-grained observations of individual Web transactions, and share this with each other. A client joining WebProfiler performs a one-time software install, which includes a Web browser plug-in and a network service. Fig. 1 shows WebProfiler's main components.

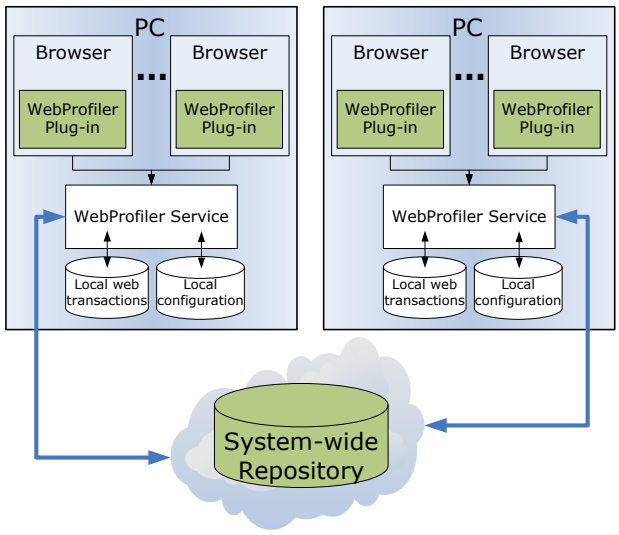

Figure 1: WebProfiler architecture

3.1 Browser plug-in

To perform fine-grained diagnosis of Web failures, we need: (a) information about user actions, such as when the user attempts to access a Web page and when she hits "Stop", and (b) details about the Web transaction, such as what IP address the request went to, how long it took and what errors, if any, were returned (a detailed list is in § 5). Popular browsers export rich APIs for extensibility [1, 5], which allows us to build a plug-in that can perform this event monitoring.

Our plug-in automatically loads when the browser is launched and runs within the browser process. It attaches a separate listener to each tab or window the user opens. The listener monitors browser events of interest, including successful transactions and any failures that occur during browsing. This platform allows broad definitions of failure — we consider Web page loads that take too long to be failures (see § 6.1.2).

3.2 WebProfiler service

Information about each Web transaction is sent over a local channel from the plug-in to the WebProfiler service on the same machine. The daemon service aggregates information from multiple browser instances on the same machine. It records Web transaction data in a local data store on the hard drive. When the system-wide repository is available, the service periodically sends this data to it. The local store allows the service to continue recording information during network partitions, when the system-wide repository is unavailable.

The service also periodically collects Web browser and network configuration settings, such as HTTP proxy settings and DNS server addresses. This is needed to correlate an observed failure with the network configuration at the time.

3.3 System-wide repository

The system-wide repository collects information sent by WebProfiler services running on different clients. It contains an aggregation of data from several clients: machine configurations, and the results of all attempted Web transactions (although, with a large number of participating clients, this could be sampled). These records are then looked up by our diagnosis algorithm which is described in the next section.

While our design and current prototype share the full complement of information on transactions with the system-wide repository, this is not necessary. WebProfiler requires clients to only increment the aggregate success/failure counts, maintained in the system-wide repository, for network elements involved in a transaction (client, proxy, server, etc.) (see § 4.2). We implement full sharing for debugging our prototype and more detailed analysis for this paper.

For corporate intranets, where there is typically some form of centralized control, the repository can be a database server. While our diagnosis primarily targets end users, this single data store implementation is simple and allows data inspection and analysis by system administrators as well [1].

To share information across the wide area Internet, the repository can be implemented using a P2P DHT. This does not require dedicated servers and allows ad-hoc collaboration between any subset of clients. For example, there can be multiple, distinct DHT "rings", e.g., for the subnet, campus, and Internet. Each client would join the rings corresponding to its location and report relevant information into each ring.

Our prototype, described in § 5, targets failure diagnosis in an enterprise network and thus implements the system-wide repository in a centralized fashion.

4 Web Failure Diagnosis

We now present our goals in diagnosing Web failures for users and then examine our simple, collaborative, blame attribution algorithm.

4.1 Diagnosis scope

The primary customers of WebProfiler's diagnosis are *end users* and we scope our diagnosis accordingly. The network element(s) that we blame for a failure are those visible to the client and, in some cases, an alternate can be used. Thus WebProfiler focuses on problems:

- solved by changing a setting on the client (e.g., switch to a good HTTP proxy)
- specific to the client's location. E.g., if a subnet is blamed, the user can switch from the wireless network to wired Ethernet or walk to a different access point.
- at the server end. Users can go to a different CDN replica or another website (e.g., if looking for generic news), contact a help line (e.g., for a banking site), or, at the very least, avoid frustration knowing that the Web site is just down.

[1] Basic connectivity failures (e.g. subnet gateway failure) can affect both Web transactions and access to a database server. However, trivial connectivity issues can be directly diagnosed at the client without cooperation [7].

This indicates the type of diagnosis that is of primary interest to end users, and is therefore our goal in WebProfiler. Nonetheless, such diagnosis can also be of tremendous value to network administrators who cannot detect from server logs those failures that happen closer to clients. However, we cannot diagnose problems not visible to clients, such as which router is to blame for poor performance on an end-to-end path.

4.2 Diagnosis procedure

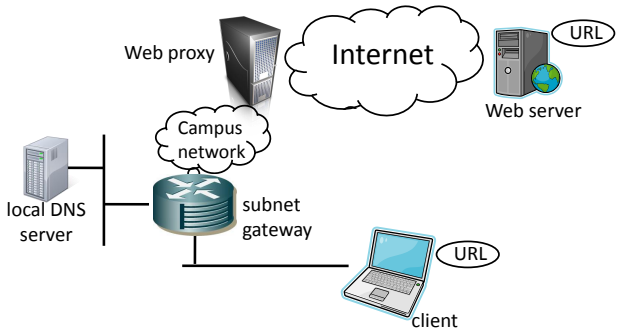

Figure 2: Some entities in typical Web transactions

Based on these goals, WebProfiler diagnoses failures at the granularity of client-visible network elements or entities such as those in Fig. 2. Let $E = \{E_1, E_2, \cdots, E_n\}$ denote the set of entities associated with a page download. For example, E might be $\{clientIP, clientSubnet, DNSserverIP, proxyIP, CDNserverIP, URL\}$. If the download succeeds, we presume that each of these entities worked fine; if it fails, any one client's perspective may not be enough to finger the culprit (see § 2.2).

We use a simple *blame attribution* algorithm outlined in Fig. 3. For each entity, E_i, there are two counts, S_i and F_i, of the successful and failed downloads, respectively, that involve entity E_i. The entity's *blame score* is $B_i = F_i/(S_i + F_i)$. Entities with an abnormally high blame score are suspects for a given failure. This blame score is compared against the distribution of blame scores for all known entities of the same functional type. For example, we would compare the blame score for proxy P1 against the distribution of scores for proxies P1, P2, $\cdots$ Pn (assuming there are n proxies in the network). We consider blame score distributions within the same functional type under the intuition that the incidence of failure that is normal for one type of entity may be abnormal for another type of entity. We recommend blame scores that are two standard deviations above the mean to be abnormally high, but this may be deployment-dependent.

An entity's blame score, while primarily reflecting its own failure rate, is also "polluted" by the failures of other entities that it is coupled with as part of page downloads. The intuition behind blame attribution is that, given a large and diverse set of observations, such incidental attribution of blame will be diminished and the true culprit will stand out. For this to be true, more complex entity relations must be considered as well. For example, Web server W1 might block requests from a particular subnet S1 . The blame score of neither W1 nor S1 might stand out, but the score of the combination of [W1, S1] will. Thus we expand the

```
// find list of suspects for failed Web transaction F
DiagnoseFailure(F)
  // find entities with abnormally high blame scores
1.   create empty set of candidate suspects S
2.   foreach entity (pair) E_i involved in transaction F
3.        retrieve success and failure counts S_i and F_i
4.        calculate blame score B_i = F_i/(S_i + F_i)
5.        retrieve distribution of blame scores B_dist for
          all entities of the same functional type as E_i
6.        if B_i is abnormally high compared to B_dist,
7.             add E_i to S
  // narrow down the list of candidate suspects S
8.   foreach suspect s_i in S
          // identify the largest superset(s) of entities
          // with abnormally high blame score(s)
9.        if s_i is contained in another suspect s_j
10.            remove s_i from S
11.  return S
```

Figure 3: WebProfiler diagnosis pseudo-code

set of entities E associated with a page download to also include *combinations of the base entities*. To limit combinatorial explosion, our prototype considers only individual entities and pairs of entities. We believe it would be rare and rather unusual for failures to be caused by triplets or larger combinations of the base entities. Our evaluation in § 6 shows that considering single entities and pairs captures most of the observed failure scenarios.

Finally, in keeping with the principle of parsimony, we strive to find the simplest explanation that is supported by the observations. That is, we lay the blame on the smallest entity with abnormally high blame scores. For example, if [W1, S1] has a high blame score, and [W1] also has a high blame score, but [S1] has a low blame score, then [W1] is the most likely suspect.

The timescale of diagnosis depends on the density of deployment. The larger the number of clients participating in WebProfiler, the shorter the timescale at which we can meaningfully compute blame scores, and the finer the time resolution of diagnosis. In our evaluation, we work with a timescale of 1000 minutes given our limited deployment.

4.3 Communication overhead

Collaborative diagnosis comes at the cost of network communication overhead in pooling together observations from diverse clients. However, as previously mentioned above and in § 3.3, our blame attribution algorithm needs to share only the success and failure counts for entities involved in Web transactions. Thus once the list of entities is populated, each Web transaction only increments counters. This incrementing can be done in batches to further reduce overhead. Also, clients can sample instead of reporting all their transactions. Finally, some observations do not need to be shared beyond limited scopes. For instance, the blame score of a subnet gateway is not relevant beyond that subnet.

5 Implementation

Component	Lines of C# code
browser plug-in	737
network service	1351
browser/service shared library	1088
diagnosis algorithm	1967

Table 1: Implementation size (without code comments)

We have implemented a prototype of the WebProfiler architecture on the Microsoft Windows platform. The source code size is in Table 1. In this section, we provide details on our implementations of the browser plug-in, network service and system-wide repository, and point out the prototype's limitations.

5.1 Browser plug-in

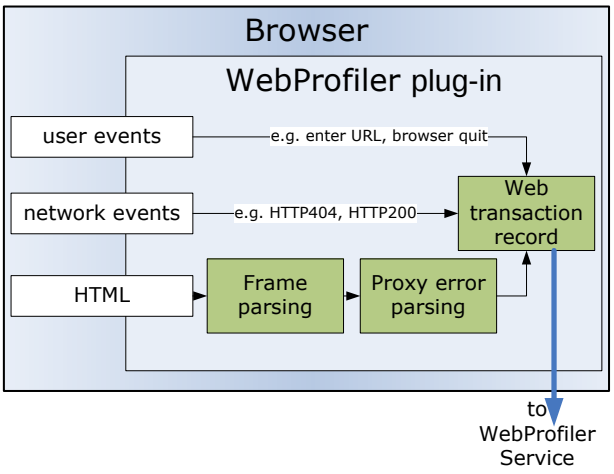

Figure 4: Browser plug-in implementation

Figure 4 shows the major components of our plug-in, which is implemented as a Browser Helper Object [1] in C#, and runs within Internet Explorer 7. The plug-in obtains several user and network events [2] from IE-7. User events such as opening a new tab/window, navigating to a new URL and closing a tab are used to delineate separate browsing sessions. Network events include completion of a Web page load, HTTP redirections and any HTTP or transport-level errors.

The plug-in also parses the HTML of downloaded Web pages to understand the frame structure, and extracts the frame URLs to distinguish between frame-level errors and top-level page errors. In addition, it detects when the returned HTML is actually a formatted error message from an HTTP proxy, and extracts information about the problem the proxy encountered in communicating with the Web site.

For every Web transaction, the plug-in constructs a record, which it then passes to the network service via the .NET Remoting framework [6]. This provides convenient interprocess communication primitives, while also taking care of data marshaling and unmarshaling. In particular, the following information for every transaction is sent to the service:

- date & time; total download time; URL requested; URL received; success or HTTP error code; premature exit by user?; IP address of HTTP proxy used (if any); error code returned by HTTP proxy (if any)

At any time, there can be multiple plug-ins running on the same client machine, either because the user has launched multiple IE-7 windows or tabs, or due to IE-7 having been launched by multiple users on a server OS. All these plug-ins communicate with a single WebProfiler service instance running on the client.

5.2 WebProfiler service

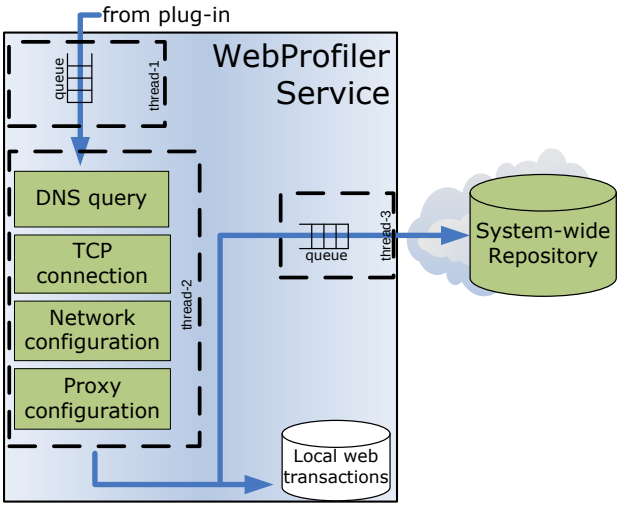

Figure 5: WebProfiler service implementation

Fig. 5 shows the major components of our service prototype. It is implemented as a Windows Service and is automatically started and shut down by the OS at system boot and halt. It runs with the same privileges as other network services on the client machine.

The service gets the client's network settings via the System.Net.NetworkInformation.NetworkChange .NET class which also alerts the service to any setting changes. The service determines the use of a socket-level proxy by detecting the presence of configuration files for the Microsoft ISA firewall in the local filesystem. It is alerted to any changes to these files via the System.IO. FileSystemWatcher .NET class.

For every Web transaction record it receives, the service also needs detailed information about the outcome of DNS and TCP attempts that IE-7 made. Unfortunately, we have not yet been able to determine how to grab DNS and TCP error codes from the application level WinInet library that IE-7 uses to fetch Web pages. As a workaround, our service re-issues DNS queries for the Web server name and the HTTP proxy

name for all Web transaction records. It is likely that these second attempts, performed immediately after the primary ones by the browser, will suffer the same fate and will be served out of the DNS cache on the client. Our service attempts a TCP connection for every failed transaction with a non-HTTP failure code from IE-7. For sending DNS and TCP messages we use the Dns.GetHostEntry and Sockets.TcpClient classes in System.Net.

5.3 Local database

For every Web transaction record the service attaches the following additional information:

- hostnames for {client, socket proxy}; IPv4 and IPv6 addresses for {client, gateway, WINS server, DNS server}; DNS A-record and AAAA-record lookup results and error codes for {Web site, HTTP proxy, socket proxy}; TCP connection error codes for {Web site, HTTP proxy}

This augmented Web transaction record is then stored locally in a database file using the .NET OLE DB class. The service also submits the record to the system-wide repository. To avoid blocking the entire service when interacting with the local database or the system-wide repository, we employ a combination of queues, timers and threads shown in Fig 5. Thread-1 interacts with (multiple) plug-ins on the client and adds each incoming Web transaction record to a local FIFO queue. Every 5 seconds, a timer wakes thread-2 to service this queue by adding DNS, TCP, network settings and socket proxy configuration to the record and writing it to the local database file. It also adds the record to a queue for the system-wide repository. Every 10 seconds, another timer wakes thread-3 to service that queue and upload records to the repository. If the repository is unreachable, this thread backs off exponentially up to a maximum of 5000 seconds. These settings of 5, 10 and 5000 seconds are engineering choices that worked well in our deployment and evaluation in an enterprise network. Each record in the local database is flagged once it has been uploaded to the system-wide repository. After a client reboot, only unflagged records are added to the queue for the system-wide repository.

5.4 Central database

The system-wide repository is implemented on a server running Microsoft SQL Server 2005. This choice is well-suited for the operating scenario our prototype targets, namely better Web failure diagnosis in enterprise networks. The central database also allows administrators to manually troubleshoot problems, and yet does not alter user privacy since router or proxy logs contain similar information for successful transactions.

Due to the limited size of our evaluation testbed and the need to produce detailed analysis for the purposes of this paper, our implementation does not include the communication overhead reduction techniques from § 4.3 and instead shares the full complement of information.

5.5 Enterprise network complexities

Our implementation considers several aspects, specific to enterprise networks, that add to the complexity of diagnosing web failures.

Corporate networks typically have firewalls at network edges to segregate intranet traffic from Internet traffic. IPSec authentication and encryption can further protect intranet traffic from un-authorized hosts and eavesdroppers. Hosts in the DMZ (de-militarized zone) between the firewalls and the public Internet, may be used for various purposes, including application testing.

In the intranet, which is segregated from the extranet, enterprise networks may use internal DNS servers. These servers resolve hostnames of internal machines only. Some may use special dynamic DNS protocols or WINS (Windows Internet Name Service) to allow internal machines with dynamic IP addresses to still have hostnames that persist across IP address changes.

HTTP proxies are typically deployed at the network edges to allow employees to access public Web content. These proxies provide performance advantages by caching popular content, and increase security by filtering malicious Internet content. Web browsers such as IE-7 and Firefox 2 automatically detect a proxy via the WPAD [8] protocol, and will tunnel all requests for external Web content to the proxy. Some commercial products, such as Microsoft ISA (Internet Security and Acceleration), also provide socket proxy functionality. This allows non-HTTP applications, such as SSH, to access hosts on the Internet.

Each proxy hostname may resolve via WINS to one of several IP addresses, each corresponding to a separate proxy server. This provides load balancing and fault tolerance. Furthermore, large corporate intranets can stretch across the globe over private or leased links or tunnels. There may be proxies at the edge of the network in each location or city that the intranet covers. This provides users in New York to use the New York proxy that peers locally with US ISPs to access the Internet, instead of traveling over leased links to the London headquarters and then entering the Internet. These proxies can also be used to access intranet websites and thereby take advantage of caching.

WebProfiler handles these aspects of enterprise networks. Our plug-in collects information at the application layer so that we can operate even in IPSec environments. During URL resolution, it detects the use of internal hostname resolution such as WINS and captures the result or any error codes. The plug-in is able to detect which Web transactions went directly to the Web server or were tunneled through a proxy. It captures the name of the HTTP proxy, and also the specific IP address of the proxy used in that transaction. The WebProfiler service detects socket proxies, in case of tunneling underneath the browser.

5.6 Limitations

Although fully functional, our current prototype has certain limitations. If a Web server returns an error page in HTML, but without setting the corresponding HTTP error code, then IE-7 does not register an error, and, as a result, neither does our plug-in. Note that this is different from the case when a proxy returns a formatted error message; our plug-in successfully parses that because it contains an HTTP error code. In addition, certain Web pages use HTTP redirects in such a way that it appears as though the original requested page did not load completely. For example, `http://office.microsoft.com` redirects to `http://office.microsoft.com/en-us/default.aspx`, which then gets redirected again to itself. The problem is that the two redirects cause IE-7 to think a generic error has occurred. We encountered 5 such Web pages during our evaluation. The evaluation results we present in the next section do not include transactions for these 5 URLs.

Even though our implementation is able to detect failures within sub-frames and for individual images, for evaluation purposes we only consider failures for the top-level frames of a Web page.

The prototype detects the use of a socket-level proxy by looking for the Microsoft ISA Firewall Client on the client machine. It can also parse HTML error pages returned from HTTP proxies that are running Microsoft ISA Server. While we have chosen to interface our prototype with these particular commercial proxy/firewall products, given the small amount of code that is specific to this choice, it is straightforward to extend it to include other alternatives.

Lastly, our prototype currently only works with IE-7 on the Windows platform, including XP, Server 2003, Vista and Server 2008. However, we believe it would be straightforward to implement it for other

common browsers. For example, Firefox exposes several rich APIs for extensions [5], and there are known ways to access browser events in Safari.

6 Evaluation

We now present the experimental evaluation of our WebProfiler prototype. We emulate users in different parts of the network by placing clients in different locations and subnets, and using a wide variety of proxies. We instrument each client to randomly browse to a pre-determined set of URLs. As noted in § 3.3, the clients not only share the success and failure counts needed for WebProfiler's diagnosis algorithm, but in fact share full information about their Web accesses to facilitate more detailed analysis for the purposes of this paper.

6.1 Experimental methodology

6.1.1 Clients

Client	OS	Location	Subnet	Proxy
client01	Win2003	Redmond	172.31	Singapore
client02	Win2003	Redmond	172.31	SouthAfrica
client03	Win2003	Redmond	172.31	Redmond1
client04	Win2003	Redmond	172.31	Redmond2
client05	Win2003	Redmond	172.31	Redmond3
client06	Win2003	Redmond	172.31	Redmond3
client07	Win2003	Redmond	172.31	Redmond2
client08	Win2003	Redmond	157.56	Singapore
client09	Win2003	Redmond	157.56	SouthAfrica
client10	Win2003	Redmond	157.56	Redmond2
client11	Win2003	Redmond	157.56	Redmond3
client12	Win2003	Redmond	157.56	Europe
client13	Win2003	Redmond	157.56	Redmond3
client14	Win2003	Redmond	157.56	Redmond2
client15	Win2008	Redmond	172.31	Europe
client16	Win2008	Redmond	172.31	Singapore
client17	Win2003	Redmond	172.31	Redmond3
client18	Win2003	Bangalore	65.52	SouthAfrica
client19	Vista	Bangalore	65.52	Singapore
client20	Vista	Bangalore	65.52	Redmond2
client21	WinXP	Bangalore	65.52	Redmond3
client22	Vista	Bangalore	65.52	Europe
client23	Vista	Bangalore	65.52	Singapore
client24	WinXP	Redmond	external	
client25	WinXP	Redmond	external	

Table 2: Clients used in experiments

Table 2 summarizes the configuration of 25 clients that we use in our evaluation. They run a variety of Microsoft Windows operating systems and are dedicated for the purpose of the experiments. They are spread across three different office buildings in Redmond, WA, and Bangalore, India. Each client is connected via wired Ethernet to one of four subnets. They are part of the Microsoft corporate intranet that stretches across the globe over private or leased links and tunnels. Client24 and Client25 are in the external DMZ (de-militarized zone) and thus do not use proxies to access the Internet, but are unable to access internal websites.

Each client on the three internal subnets was configured to use a specific HTTP proxy and socket proxy to access the Internet. As shown in Table 2, we picked proxies in four different locations. Redmond offers several proxy servers with different configurations and connections to the Internet, of which we use three separate ones. While a typical client would automatically select the proxy closest to its network location, we intentionally use a diverse set of proxies in an attempt to emulate the diversity that would be naturally

present in a large-scale deployment of WebProfiler.

6.1.2 User emulation

1. Copy workload (list of URLs) into main memory
2. Seed random number generator
3. Randomly shuffle URLs using Fisher-Yates [3]
4. Launch IE-7 for next URL
5. Sleep for 20 seconds
6. Signal IE-7 to quit (also clears the cache)
7. Sleep for 60 seconds
8. Go back to step 4 unless there are no more URLs
9. Go back to step 1

Figure 6: Pseudo-code for browsing process

To emulate the Web browsing behavior of users, we automate the IE-7 browser using the InternetExplorer API exposed by shDocVw.dll. The pseudo-code for our 148 lines of C# code is shown in Fig. 6. Each client executes the browsing process indefinitely, until we terminate the experiment. We pick 20 seconds as an extremely conservative estimate of how long a user would wait for a Web page to load. We allow roughly 80 seconds between successive Web page accesses by a client. Although this process may not necessarily model how typical users browse the Web, we believe it is sufficient for evaluating our prototype.

Before starting an experiment, we pre-configure IE-7 on each client. As mentioned earlier, we specify a proxy on our internal subnets. Furthermore, we disable Java, set the Home Page to blank and turn off most user prompts. We also enable an IE-7 option called "Empty Temporary Internet Files folder when browser is closed" to force each Web access to be served from the network.

6.1.3 URL workload

Type	URLs	Rate
Nielsen 1-50	50	1x
Nielsen 3858-3958	50	1x
Internal	25	1x
Other	5	1x
Control (internal)	4	12x
Control (external)	1	12x

Table 3: URL workload used in evaluation

The 135 unique URLs we use for the user emulation process are summarized in Table 3. The majority of these come from the Nielsen/NetRatings subdomain list, a compilation of the Web server names that carry the

most visited Web pages, for users in North America in February 2007. We pick the top 50 subdomains, which include sites such as `http://www.yahoo.com/`, `https://login.yahoo.com/`, and `http://www.myspace.com`. We also pick 50 entries from the bottom of the list, around rank 4000, which include sites such as `http://channels.isp.netscape.com/` and `http://kevxml2adsl.verizon.net/`. We intentionally avoid Web sites with adult content to avoid violating corporate policies. As noted in § 5.6, we also do not include 5 URLs due to the problematic way they do HTTP redirects. Furthermore, we ignore extraneous accesses generated by pop-up advertisements. We manually select 25 URLs of internal corporate Web sites that are located all across our corporate network, including those in Redmond, Mountain View, Canada, UK, Italy, Middle East, India, and Australia. The workload also includes 5 more public URLs that the authors often visit.

We pick this URL workload with two goals in mind. First, we wish to include popular Web sites that most users visit. Second, some fraction of user Web activity occurs on sites that are not as universally known, but rather reflect individual tastes. While we do not claim to accurately model the list of Web sites most users visit, we believe this is a reasonable workload for evaluating Web failure diagnosis.

6.1.4 Control failures for validation

We also run one external and four internal Web servers that are accessed 12 times more frequently. We manually induced specific failures to evaluate the veracity of our diagnosis algorithm. The four internal servers are located in Bangalore. In addition to not being accessible by the external clients 24 and 25, these four internal servers are configured to:

1. be highly available
2. drop 20% of internal requests [2].
3. drop all requests from Redmond2 proxy
4. drop all requests from client18

We omit discussion and detailed results for the external control server due to experimental error in manually inducing its faults.

6.1.5 Diagnosis parameters

We briefly discuss the parameter settings for WebProfiler in our evaluation. These settings are influenced heavily by the modest scale of our experiment. During a 7-day period, each client would access each URL about 39 times. So, for each entity, whether a base entity or a pair, we should have at least 39 observations. Of course, many entities (e.g., individual URLs, pairs of [proxy,server]) would have many more observations. In general, we insist on there being at least 20 observations (successes or failures) for an entity during the period of interest, for us to be able to compute the blame score for that entity in a meaningful way.

This sparsity of the observations means that using too long a period for failure analysis (e.g., 7 days) would only capture persistent failures, whereas too short a period (e.g., a few minutes) would leave us with too few observations to be able to compute the blame score for most entities. In view of this, we use an intermediate period of 1000 minutes, which would yield a sufficient number of observations, not only for individual entities, such as URLs, but also for pairs such as [subnet,URL]. Of course, in a large-scale

[2]From incoming requests, it randomly picks a source IP address with 20% probability and drops all TCP SYN packets from it for one minute, including all retries of the original connection attempt.

deployment, there may well be sufficient density of observations to permit diagnosis even on a timescale of minutes. Regardless of timescale, the same diagnosis procedure applies.

In terms of the entities for the purposes of diagnosis, we consider the URL, Web site hostname, client hostname, and IP addresses for each of gateway, DNS server, and HTTP proxy, along with their pairs.

§ 4.2 recommends using two standard deviations above the mean when comparing a blame score for an entity to the scores of other entities of the same type, to determine if it is abnormally high. Given the limited deployment of our evaluation, some entity types (such as gateway or DNS server) are small in number, and thus we cannot effectively employ this heuristic. We instead do a strict comparison of the blame scores across all candidate suspects, and return the highest two.

6.2 Results

Start	End	Success	Fail
2007-09-21 00:47	2007-09-26 19:37	99,152	50,799
2007-09-28 01:53	2007-10-04 22:18	139,722	44,839

Table 4: Measurement periods (UTC)

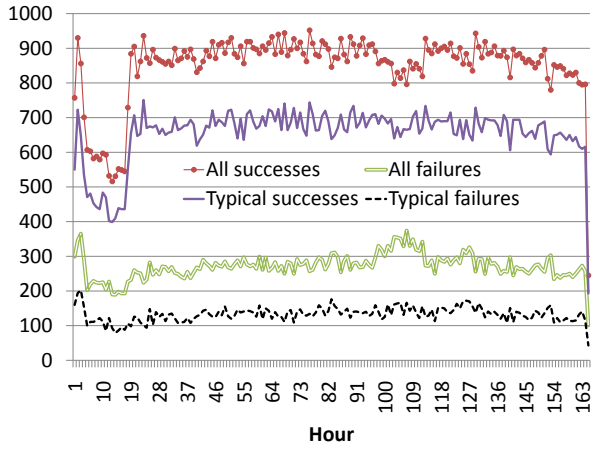

Figure 7: Failures over time (aggregated by hour)

We now present our experimental results. Table 4 shows our measurement periods, as well as the total number of Web transaction successes and failures, as recorded on all the clients. We present results only from the second, more recent period (the results are similar). Note that the high failure rate is due to the manually induced failures for the 5 websites we control.

6.2.1 Web access failures

Fig. 7 shows the number of successful and failed client attempts over time. "Typical successes" means those not including the 5 URLs accessed with 12x rate, and not those URLs that always fail for all clients. "Typical failures" similarly indicates the amount of failures that users will typically see in our network. The failure rate remains fairly constant, except for around the experiment's tenth hour when a maintenance interval caused some clients to reboot.

Fig. 8 shows the total number of attempted accesses for each client, broken down by successes and failures. Two clients, "client20" and "client22" have a very low access count. Upon manual inspection of our experiment logs, we discovered that both suffered a permanent failure as a result of the maintenance interval.

Fig. 9 shows the number of failures per unique URL visited. They are all quite similar, except for the first 5 URLs, which are those that have a 12x access rate. Lastly, Fig. 10 shows the cumulative number of URLs ordered by failure rate. Very few URL accesses experience failure more than around 45% of the time.

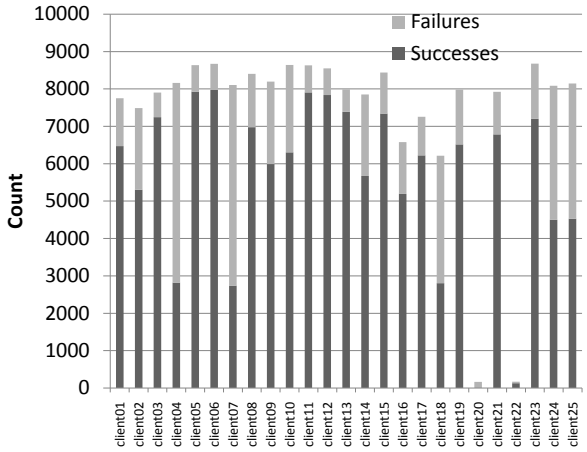

Figure 8: Web accesses per client

6.2.2 Blame entities

Table 5 shows the efficacy of failure diagnosis for different levels of collaboration. At one extreme is an individual client performing diagnosis by itself (i.e., no collaboration). At the other extreme is collaboration among the entire set of 25 clients. We also consider intermediate points — within a subnet, region, etc. Each collaboration level offers a different degree of diversity in terms of vantage points. So, diagnosis within a subnet has only one gateway to blame, but across all clients there are multiple gateways. More vantage points can increase the number of entities, but not all combinations of entities are naturally exercised by the clients. The "total" column shows the total number of blame entities, including their pairs at each collaboration level. Note that the HTTP proxies column corresponds to the (multiple) IP addresses of the specific proxy, which exceeds the number of proxies in Table 2.

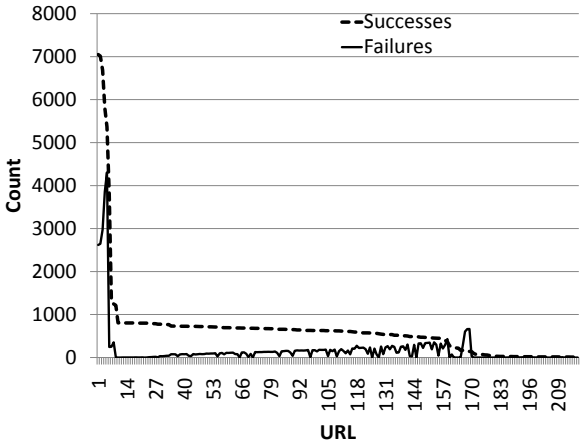

Figure 9: Failures per URL

6.2.3 Diagnosis accuracy for controlled failures

We check the accuracy of our diagnosis algorithm with the 4 types of controlled failures from Section 6.1.4. In these specific cases we know the true cause (the client visible entities to blame for the failure), since we manually introduced the failures. For example, for the first type of controlled failure, where external clients cannot access the website, the blame should be laid on the combination of external clients and the website, because accesses from other entities or combination of entities results in high success rates.

Table 6 presents our findings for each of these URLs, and for each level of collaboration. We see that when using WebProfiler to collaborate across **all** 25 clients, the true cause is accurately identified as the top suspect for 5852 failures (3379 + 2144 + 329). For 3652 failures, it is the second most likely suspect (923 + 910 + 911 + 908). In contrast, when not using any collaborative diagnosis (**client** lines), the true cause is identified as the top suspect for only 2639 failures, and not found for the remaining 6890 failures (908 + 1666 + 3076 + 1240). Note that we consider here only those failures caused by our manual intervention.

Thus, for the controlled failures where we can measure our prototype's accuracy, *our diagnosis system accurately diagnoses 3.6 times as many failures as a single client could* ((5852 + 3652) / 2639). WebProfiler accurately diagnoses over 99% of these failures (for "**all**" lines : sum of first two columns divided by sum of all three columns).

6.2.4 Diagnosis accuracy for all URLs

When all failures seen by our clients (for all URLs in Table 3) are diagnosed, we find that the average number of suspects reported is quite low at 1.83. For most failures, our algorithm finds 1 or 2 suspects to blame. This result is similar over a 1000 minute timescale and the entire 7 day duration.

When we attempt blame attribution with any single client alone, the algorithm tends to blame just the URL or the Web site, since, from that client's perspective, the Web site just seems to be down. The other entities work for other Web sites, and have a lower blame score. It is only the information from other clients that can tell us whether the Web site is indeed available. So, even though our algorithm still produces a small number of suspects when diagnosing from any one client alone, it is often incorrect.

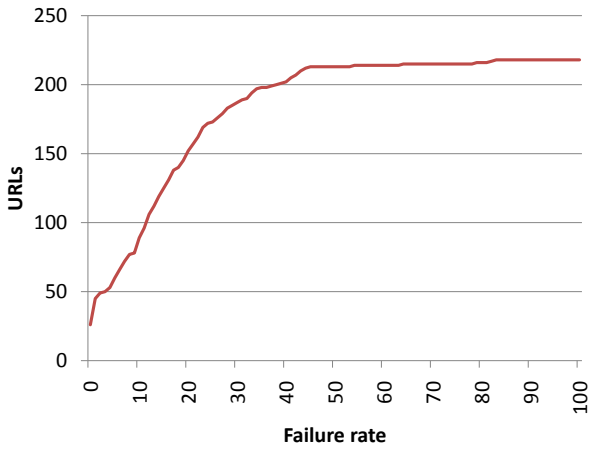

Figure 10: Cumulative number of URLs by failure rate

7 Extending WebProfiler to the Internet

Certain characteristics of our current WebProfiler prototype, such as the centralized database server, make it better suited for deployment in an enterprise setting, where IT resources, which users have to trust anyway, are available to host the shared database. In this section, we discuss the applicability of and the new demands placed on our collaborative approach in a wide-area setting.

7.1 Decentralized Data Storage

If participating clients are spread across multiple ISPs, a dedicated data storage infrastructure may not be available. A viable alternative is a decentralized, peer-to-peer design, where client hosts contribute storage and bandwidth to cooperatively implement the system-wide repository and support failure diagnosis. WebProfiler clients could utilize existing Distributed Hash Table (DHT) implementations to hold information (e.g., blame score) indexed by the identity of the entities involved.

Several DHT implementations exist (e.g., Chord, Kademlia, Pastry, Tapestry), each with its strengths and drawbacks. We particularly care about locality, for instance, to ensure that subnet-specific information remains within the subnet. One possibility is to use SkipNet [15], which provides controlled data placement to ensure locality. An alternative is to have multiple, distinct DHT "rings", e.g., at the level of the subnet, campus, and Internet. Each client would join the rings corresponding to its location and report relevant information into each ring.

Of course, realizing the system-wide repository over a peer-to-peer system entails other complications, such as dealing with nodes with relatively low availability, and with connectivity challenges due to NATs and firewalls. However, prior work on DHT replication and routing strategies in the face of high node churn could be leveraged to address these issues.

View	clnts	gwys	DNS srvr IPs	HTTP proxy IPs	Web srvr IPs	URLs	total
client(avg)	1	1	1	6	141	194	30734
subnet 172	10	1	1	29	157	250	56754
subnet 157	7	1	1	27	164	273	61235
subnet 65	6	1	1	27	152	249	52548
Redmond	17	2	1	29	171	311	77963
internal	23	3	2	29	172	321	84796
external	2	1	2	0	147	217	34096
all	25	4	4	29	173	334	90958

Table 5: Number of blame entities per view

7.2 Security, Privacy, and Incentives

Given our focus on enterprise networks, we have not described security and privacy issues. Corporations already watch user activities, and there is less concern about untrusted users providing incorrect information to WebProfiler. Such issues would have to be addressed in a wide-area scenario though. In particular, client observations should be anonymized by the service before being sent to the system-wide repository. For instance, for data shared within the campus ring, the client IP would be concealed, whereas for data shared across the Internet-wide ring, even the subnet should not be revealed.

Furthermore, the system should be resilient to malicious services that might send falsified records of client activity, attempting to throw off failure diagnosis. Given the volume of shared data, a voting scheme could arguably be devised to weed out any non-consistent information. In addition, no incentives are currently given to clients to contribute. As a result, a selfish client could just utilize the system for diagnosing its own failures, yet never contribute any information itself. However, there already exist effective techniques to enforce cooperation in a wide-area peer-to-peer storage system [11].

8 Related Work

8.1 Measurement studies

Many prior measurement studies of Internet performance and failures have considered either Web transactions, or individual facets of the wide-area network, such as TCP performance, routing and DNS. Given the large body of work, we provide only a brief overview.

WebProfiler builds on our prior measurement study [18], which characterized the nature of Web failures based on accesses made from 134 diverse client hosts. The prior study uses these measurements to classify the types of failures (e.g., specific types of TCP failures, correlations with BGP routing fluctuations, etc.) and quantify the rate of failures at specific locations (e.g., are some servers worse than others?).

Dahlin *et al.* [12] analyze Web service availability using proxy logs and traceroutes. They report an unavailability rate of 0.5-2.0%, with stub network and interior network failures contributing significantly. Gummadi *et al.* [14] report that the majority (60%) of failures on paths to broadband hosts occur on the last hop or the end host itself. Other work [24, 13] has considered the stability of end-to-end routes.

Several studies have focused on DNS because of its importance. Pang *et al.* [19] report high availability for a broad set of local and authoritative name servers encountered in Akamai logs. In addition, CoDNS [20] studies the performance of DNS queries issued by a set of PlanetLab nodes for each other's names, and observes that many failures can be attributed to overload at the local DNS server.

8.2 Diagnosing web failures

Prior techniques for automatically diagnosing Web failures include local diagnosis of a client's network components, Web or proxy server log analysis and packet trace analysis.

The Network Diagnostics Framework [7] in Windows Vista troubleshoots simple network problems. It comprises a collection of Helper Classes, each of which contains the logic required to evaluate the health of its respective local network component (e.g. DNS, Ethernet interface), and repair it. For example, it can detect and attempt to repair a Web access failure because the Ethernet interface does not have an IP address.

Kıcıman *et al.* [16] present two algorithms that enable content providers to extract Web failure information from logs at their CDN servers. However, log based analysis cannot capture failures that prevent a request from even reaching a server.

More recently, Sherlock [9] targeted network performance problems in large enterprise networks. They monitor packet flows and probabilistically construct dependency graphs for each application within the enterprise. WebProfiler instead focuses specifically on web transactions, regardless of intranet or Internet destinations. It uses lightweight monitoring of application events to detect failures and does not require any packet inspection and is unaffected by end-to-end IPSec encryption. Due to application level support, WebProfiler does not need to rely on large numbers of similar application transactions to cluster multiple simultaneous network transactions into the related application transaction.

8.3 Multiple vantage points

The concept of using multiple vantage points to diagnose network faults has been previously explored in other contexts.

SCORE [17] does fault localization based on a shared risk model, which groups together IP links with a common underlying network component (e.g., an optical fiber). It uses bipartite matching to select the risk group(s) that explains the greatest number of IP link failures. SCORE operates on a modest number of (link) failure observations — iterative bipartite matching would be prohibitively expensive to handle potentially millions of observations, especially when new observations accrue incrementally. In contrast, our blame attribution algorithm updates the blame score for each entity independently, enabling it to scale well.

WiFiProfiler [10] uses collaborative diagnosis for wireless LAN failures and addresses challenges of ad-hoc wireless communication. Unlike our generic blame attribution, WiFiProfiler uses a rule-based approach specific to the 802.11 domain.

PeerPressure [23] uses a statistical metric based on empirical Bayesian estimation to automatically diagnose machine misconfigurations. However, it assumes there is only one misconfigured entry among all suspects, and it is limited to local misconfigurations and does not handle network failures.

Several systems rely on multiple vantage points for purposes other than failure diagnosis, e.g., for predicting the quality of connections (SPAND [22]), and network health monitoring (Keynote [4], NETI@home [21]).

9 Conclusion

We have presented WebProfiler, a collaborative, client-based system for diagnosing Web access failures. The key ideas are to leverage observations on the success or failure of individual accesses from multiple vantage points, rely on client cooperation to obtain application level information, and use a simple blame attribution algorithm to determine the cause of failures.

We have built a WebProfiler prototype on Microsoft Windows that monitors user activity at the browser level, and relevant client configuration. It stores this information in a system-wide repository, which WebProfiler queries to diagnose specific failures. Our experimental results on a controlled enterprise testbed show that WebProfiler can accurately diagnose 3.6 times as many failures than possible from a single client's perspective.

In future work, we plan to extend WebProfiler to the wider Internet, while addressing privacy concerns via anonymized routing and scalability concerns via multiple overlapping DHT layers.

References

[1] Browser Helper Objects: The Browser The Way You Want It. `http://msdn2.microsoft.com/en-us/library/Bb250436.aspx`.

[2] DWebBrowserEvents2 events. `http://msdn2.microsoft.com/en-us/library/Aa768283.aspx`.

[3] Fisher-Yates shuffle. `http://en.wikipedia.org/wiki/Fisher-Yates_shuffle`.

[4] Keynote Systems. `http://www.keynote.com`.

[5] Mozilla Knowledge Base: Getting Started With Extension Development. `http://kb.mozillazine.org/Getting_started_with_extension_development`.

[6] .NET Framework Remoting Overview. `http://msdn2.microsoft.com/en-us/library/kwdt6w2k.aspx`.

[7] Network Diagnostics Framework. `http://msdn2.microsoft.com/en-us/library/aa369892.aspx`.

[8] Web Proxy Auto-Discovery Internet Draft. `http://www3.ietf.org/proceedings/99nov/I-D/draft-ietf-wrec-wpad-01.txt`.

[9] V. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards Highly Reliable Enterprise Network Services Via Inference of Multi-level Dependencies. In *SIGCOMM'07*.

[10] R. Chandra, V. N. Padmanabhan, and M. Zhang. WiFiProfiler: Cooperative Diagnosis in Wireless LANs. In *MobiSys'06*.

[11] L. P. Cox and B. D. Noble. Samsara: Honor Among Thieves in Peer-to-Peer Storage. In *SOSP'03*.

[12] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate. End-to-End WAN Service Availability. *IEEE/ACM Transactions on Networking*, 11(2), 2003.

[13] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek. Measuring the Effects of Internet Path Faults on Reactive Routing. In *SIGMETRICS'03*.

[14] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *OSDI'04*.

[15] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *USITS'03*.

[16] E. Kıcıman, D. A. Maltz, M. Goldszmidt, and J. C. Platt. Mining Web Logs to Debug Distant Connectivity Problems. In *MineNet'06*.

[17] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. IP Fault Localization Via Risk Modeling. In *NSDI'05*.

[18] V. N. Padmanabhan, S. Ramabhadran, S. Agarwal, and J. Padhye. A Study of End-to-End Web Access Failures. In *CoNEXT'06*.

[19] J. Pang, J. Hendricks, A. Akella, R. D. Prisco, B. Maggs, and S. Seshan. Availability, Usage, and Deployment Characteristics of the Domain Name System. In *IMC'04*.

[20] K. Park, V. S. Pai, L. Peterson, and Z. Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *OSDI'04*.

[21] C. R. Simpson, Jr. and G. F. Riley. NETI@home: A Distributed Approach to Collecting End-to-End Network Performance Measurements. In *PAM'04*.

[22] M. Stemm, R. Katz, and S. Seshan. A Network Measurement Architecture for Adaptive Applications. In *INFOCOM'00*.

[23] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic Misconfiguration Troubleshooting with PeerPressure. In *OSDI'04*.

[24] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services. In *OSDI'04*.

View	# of failures where true cause is suspect#1	suspect#2	not found	Control website
all		908		
external			908	
internal				
Redmond		908		drop
subnet 65				all external
subnet 157				requests
subnet 172				
client			908	
all	3379	923	3	
external			911	drop 20% of
internal	3379	12	3	internal
Redmond	3013	915	3	requests
subnet 65			374	& all external
subnet 157	1341			requests
subnet 172			290	
client	2639		1666	
all	2144	910	22	
external			910	drop all
internal	2144		22	Redmond2
Redmond	2144		22	proxy requests
subnet 65				& all external
subnet 157	836		7	requests
subnet 172			440	
client			3076	
all	329	911		
external			911	drop all
internal	329			client18
Redmond		911		requests
subnet 65	329			& all external
subnet 157				requests
subnet 172				
client			1240	

Table 6: Diagnosis of controlled failures for different levels of collaboration